INP ENSEEIHT

TP1et2 : Mise en jambe sur les graphes en OCAML et graphes eulériens

1. Mise en jambe avec OCamlgraph

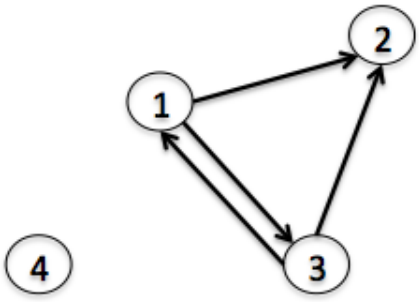
Vous allez pouvoir utiliser un module prédéfini sur les graphes. Pour utiliser les fonctions de OCamlgraph vous devez soit charger le module correspondant aux graphes :
`open Graph.Pack.Graph;;`
ou
`open Graph.Pack.Digraph;;`
pour les graphes orientés,
soit préciser le module en préfixe de vos fonctions, par exemple :
`let g = Graph.Pack.Graph.create ();;`

Une documentation en ligne est disponible à :
`http://ocamlgraph.lri.fr/doc/Pack.html`

Vous remarquerez que, contrairement au cours de OCAML où nous avons fait du fonctionnel pur, l'implémentation en OCAML des graphes fait appel à des structures mutables. De ce fait, on pourra aussi utiliser des effets de bords. On pourra en particulier utiliser l'itérateur de liste `List.iter`.
Commencez par vous familliariser avec la bibliothèque de graphes ; pour cela on vous donne la définition d'exemples de graphes non orientés (fichier `ex_no.ml`), et des fonctions pour afficher la liste des sommets et arêtes directement dans l'interprète (fichier `af.ml`). Le fichier `jbuild` est aussi fourni. Vous pouvez appeler `jbuilder utop`, puis dans l'interprète inclure les fonctions d'affichage `#use "af.ml";;`

2. Vos premiers exemples

Nous vous proposons, en exercice, de définir quelques graphes orientés, dont le suivant :



Attention d'adapter le `open ...` au début du fichier `af.ml` aux graphes orientés.

3. Une première fonction

Écrire une fonction qui prend un entier n et crée le graphe K_n.

4. Utilisation des itérateurs de graphe

Regarder l'utilisation des itérateurs `fold_edges` et `fold_vertex`.

Écrire une fonction qui prend un graphe en entrée et calcule n le nombre de sommets.

Écrire une fonction qui prend en entrée une liste de sommets V et une liste d'arêtes E et qui renvoie le graphe non orienté correspondant. Bonus : inclure une pondération dans les arêtes.

Écrire une fonction qui prend en argument un graphe et renvoie un booléen qui dit si il existe au moins un sommet de degré 0. On pourra utiliser la fonction "succ".

5. Graphes eulériens

Un graphe est eulérien (resp. semi-eulérien), ou encore peut être dessiné sans lever le crayon ni passer deux fois sur le même trait, s'il est connexe et tous ses sommets sont de degré pair (il a exactement deux sommets de degré impair).
Dans la première partie, nous allons seulement caractériser les graphes eulériens (resp. semi-eulériens). Dans la deuxième partie vous calculerez un chemin eulérien (resp. semi- eulérien).

5.1. Caractérisation d'un graphe eulérien (resp. semi-eulérien)

Pensez à définir quelques exemples de graphes pour tester vos algorithmes.

1. Programmez la fonction `est_degre_pair` qui prend un graphe et un sommet, et renvoie un booléen indiquant si le sommet est de degré pair.
2. Programmez la fonction `est_connexe`, qui prend un graphe en entrée et renvoie un booléen indiquant si le graphe est connexe. Vous pouvez utiliser le module `Mark` de la librairie OCamlgraph.
3. A l'aide des fonctions précédentes, programmer les fonctions `est_eulerien` et `est_semi_eulerien` qui vérifient si un graphe est eulérien (resp. semi-eulérien).

5.2. Calcul d'un cycle ou chemin eulérien

Pour un graphe eulérien (resp. semi-eulérien), on souhaite maintenant trouver un cycle (resp. chemin) eulérien. Ecrivez une fonction qui prend en entrée un graphe eulérien (resp. semi-eulérien) et retourne une liste de sommets correspondant à un parcours eulérien (resp. semi-eulérien).
Pour trouver une chaîne eulérienne, ou un cycle, on choisit une chaîne initiale, ou un cycle initial, puis on ajoute des cycles connectés, jusqu'à qu'on ait visité toutes les arêtes.
Quelques remarques :

— on pourra faire une copie du graphe, et éplucher ses arêtes au fur et à mesure qu'on les parcourt,
— le backtracking n'est pas nécessaire puisqu'on est sûr de trouver un parcours en avançant sur une arête (toutes les arêtes doivent être visitées).